GOOGLYEYE SERVICE

Problem Description:

Develop the "Googly Eyes" service, which exposes a single HTTP endpoint which allows a user to upload a photo and returns the same photo but modified with googly eyes in place of their eyes.

Constraints:

- Single http end point
- Varying Pupil location and size
- Able to detect multiple faces
- Robust and Performance under load

Steps followed to get googly eyes:

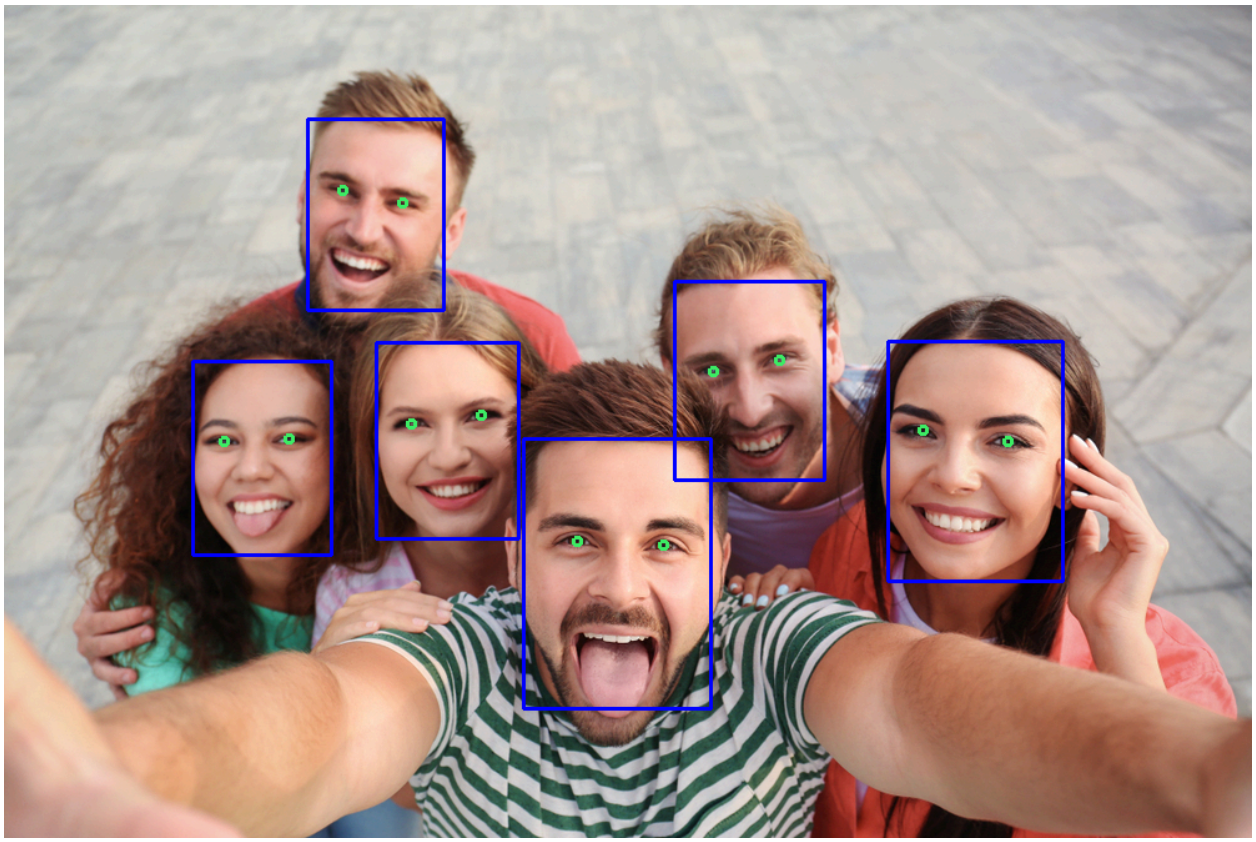
- Read the image and pass to MTCNN face detection model.
- It gives the following output for each detected face.

```
{
'box': [x, y, w, h],
 ‘confidence': 1.0,
 'keypoints': {'nose': [x, y],
 'mouth_right': [x, y],
 'right_eye': [x, y],
 'left_eye': [x, y],
  'mouth_left': [x, y]
}
```

- Only eye center is given.So, based on the size of the bounding box, I am taking 20% of height and width as the googly eye’s height and width.
- For that size, I am creating a white circle with radius as max(googly_eye_width, googly_eye_height).
- From that I am calculating randomly the size of the pupil to lay within the white circle and randomly selecting the center within the white area.

GOOGLYEYE SERVICE

Output from MTCNN:

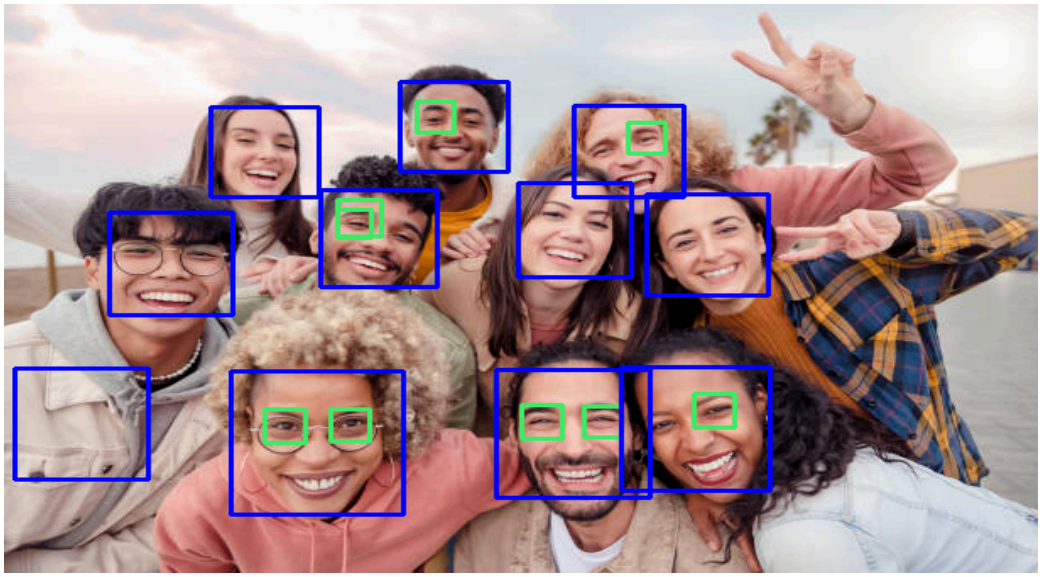


Why MTCNN?

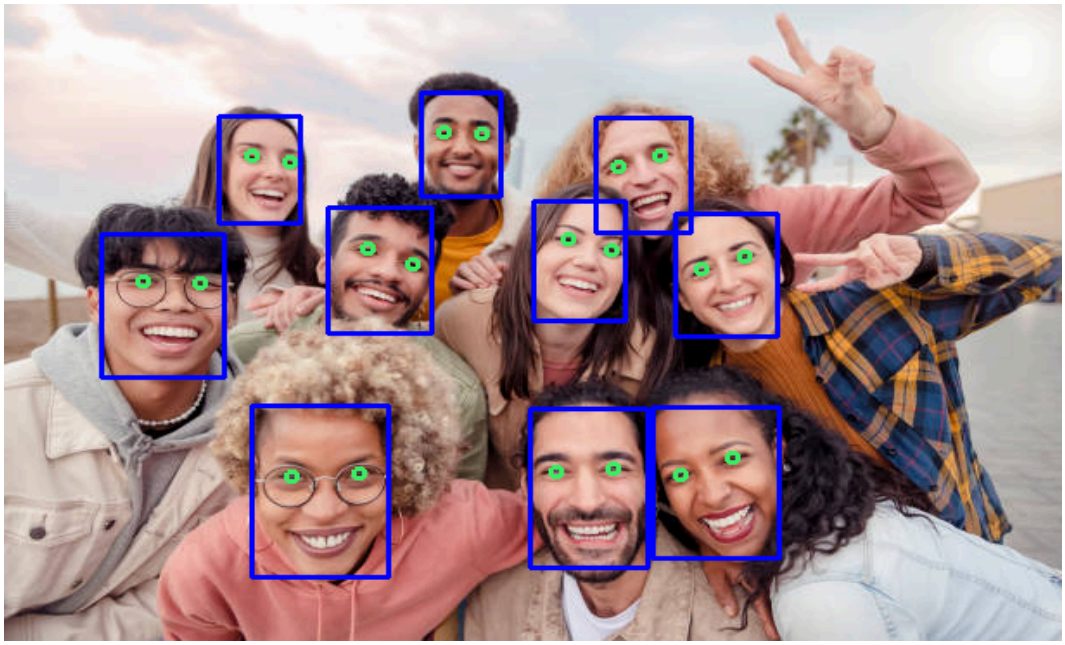
- I tried other techniques like conventional computer vision based algorithms like HaarCascade and MediaPipe.
- They were working for images with person nearby the camera, but when I was testing on edge cased, MTCNN performed really better.
- MediaPipe was unable to get landmarks if people are as far as in the images shown.

GOOGLYEYE SERVICE

Output from Haar Cascade : Some eyes are missed



Output from MTCNN:



GOOGLYEYE SERVICE

Development Environment:

- Python 3.12
- Linux
- Tested on Mac

Testing:

Pytest is used to test the endpoints.

Test 1: Load a sample image and check the response.
Test 2: Create concurrent requests using TestClient and check the load.

Packaging:

- Tox is used for virtual environment management.
- Dockerfile is also provided.

For running instructions, please go through Readme.md